

An Internet friendly transport protocol for continuous media over best effort networks

Hala ElAarag^{1,*} and Mostafa Bassiouni^{2,†}

¹*Department of Mathematics & Computer Science, Stetson University, DeLand, FL 32723, U.S.A.*

²*School of Electrical Engineering & Computer Science, University of Central Florida, Orlando, FL 32816, U.S.A.*

SUMMARY

In this paper, we design and evaluate an Internet friendly transport-level protocol (IFTP) for solving the TCP-friendly problem. IFTP has two modes of operation. In the standard mode, the IFTP connection faithfully emulates the behaviour of TCP in order to roughly obtain a bandwidth equal to that of a TCP connection. In the extended mode, a simple modification is used to grant QoS-differentiated services to selected connections. Connections running in the extended mode can get enhanced bandwidth while still emulating the general behaviour of TCP. We develop an analytical model for the congestion control mechanism of IFTP. We also derive analytically the amount of bandwidth that IFTP may be able to claim from TCP in ideal and non-ideal environments. We evaluate IFTP through simulation and prove its TCP friendliness as well as provide performance results on some of the important metrics such as packet delay, delay jitter, packet loss and link utilization. Copyright © 2002 John Wiley & Sons, Ltd.

KEY WORDS: congestion control; TCP-friendly protocols; best effort networks; media streaming; differentiated services; network simulation; Internet health

1. INTRODUCTION

Fairness is among the most important properties of data flows in the Internet. Fairness implies that whenever there is congestion at a bottleneck, each flow going through the bottleneck gets a fair share of the available bandwidth. TCP flows achieve this fairness by using congestion control mechanisms [1] that adapt TCP's throughput as a function of packet loss [2]. Some popular transport protocols, e.g. UDP, used for the transfer of multimedia applications do not incorporate any end-to-end congestion control. When TCP and UDP flows share a congested link, the TCP flows reduce their sending rate in response to congestion, while UDP flows uncooperatively use the available bandwidth.

*Correspondence to: H. ElAarag, Department of Mathematics & Computer Science, Stetson University, DeLand, FL 32723, U.S.A.

†E-mail: helaarag@stetson.edu

‡E-mail: bassi@cs.ucf.edu

Contract/grant sponsor: NSF; contract/grant number: 0086251

Contract/grant sponsor: ARO

Received 3 November 2001

Revised 24 September 2002

Accepted 14 October 2002

Multimedia applications (audio and video) are explosively growing in the Internet. In order to co-exist with current TCP-based applications in a healthy Internet environment, it is important that these multimedia applications incorporate some form of congestion control mechanism. This mechanism should be ‘TCP-friendly’ in the sense that if an application shares a bottleneck link with a TCP connection then both connections receive a fair share of bandwidth.

In this paper, we develop a congestion control algorithm for best-effort flows. The algorithm controls the sending rate in a manner that is roughly equivalent to TCP. The transmission rate is determined, based on an analytical model that emulates TCP throughout the lifetime of a connection. It takes into consideration network conditions such as round trip times and packet losses. Our protocol then computes a TCP-friendly sending rate by adjusting the inter-packet gap. The protocol is easy to implement, has little overhead, and guarantees fairness by preventing non-TCP flows from usurping the Internet. Furthermore, a scale factor can be easily incorporated into the logic of IFTP to adapt the scheme to the individual needs of certain best-effort flows. With this simple modification, a best-effort flow is allowed to get a bandwidth exceeding that of its TCP counterpart (e.g. one and half or two times) but will still behave in a friendly manner and reduce its sending rate at times of congestion. This improves the capability of IFTP to provide better QoS-differentiated services to selected traffic flows.

The rest of the paper is organized as follows. Section 2 presents some of the previous work that addressed the TCP-friendliness problem and proposed solutions. In Section 3, we present the analytical model which is the basis of the TCP-friendly protocol proposed in this paper. Specifically, we derive the formulas to calculate the inter-packet gap for IFTP. We discuss the implementation of IFTP in Section 4 and provide pseudocode for the algorithm. In Section 5, we calculate the bandwidth of IFTP in three cases: ideal, non-ideal and worst-case scenarios. Performance evaluation of the IFTP protocol using simulation tests is presented in Section 6. Finally, Section 7 gives conclusions of the paper.

2. RELATED WORK

Research in the TCP-friendly area is relatively new in the literature. Floyd discussed the problem in her work with Fall [3] and Mahdavi [4]. Some researchers proposed solutions to the problem. This included the use of differentiated services [5], which use a pricing scheme to control sharing. Another approach suggests the use of per-flow scheduling mechanisms that separately regulate the bandwidth used by each best-effort flow [6]. Legout and Biersack in Reference [7] suggested a solution to the problem by using a fair scheduler network. They also proposed a TCP-friendly protocol [8] to multicast audio/video content to a large number of heterogeneous receivers. In References [9,10], Sisalem *et al.* suggested protocols based on the end-to-end real time transport protocol (RTP). Rejaie *et al.* [11] proposed the RAP protocol. It is a rate-based congestion control mechanism that employs an additive increase multiplicative decrease (AIMD) algorithm. They used RAP as the underlying layer for streaming controlled video playback in Reference [12].

Based on the classification given in Reference [13], the IFTP protocol proposed in this paper is a rate-based, unicast, single-rate, end-to-end protocol. This means that IFTP achieves TCP-friendliness by dynamically adapting its transmission rate according to feedback mechanism that indicates congestion. It is a unicast transport protocol and hence confined to be a single rate scheme. IFTP is end to end in the sense that it does not require routers to provide any additional

mechanisms to support it. Widomer *et al.* [13] classified four protocols to be unicast, single rate, and rate based. They are RAP [11], LDA + [14], TFRC [15] and TEAR[16]. LDA + relies solely on the real-time transport control protocol (RTCP) feedback messages provided by RTP. TFRC adjusts its sending rate based on a complex TCP equation to calculate TCP throughput as a function of packet loss rate. TEAR is a hybrid protocol that combines aspects of window-based and rate-based congestion control. TEAR receivers calculate a fair receive rate which is sent back to the sender, who then adjusts the sending rate. The closest to IFTP would be RAP. However, RAP, as we mentioned earlier, uses AIMD mechanism to emulate TCP. AIMD is a steady-state model that can be reached in networks with low loss rates [4]. This means that RAP emulates TCP only in the steady state. Thus, only long file transfers suffering low loss rates can benefit from RAP. IFTP on the other hand mimics TCP in all its stages. This makes IFTP suitable for both short and long file transfers. In addition, IFTP has a simple scale-factor extension that enables the protocol to adapt to the needs of best-effort flows requiring more bandwidth than their TCP counterparts. This makes IFTP more capable of providing better QoS-differentiated services to selected best-effort flows. Also IFTP would perform better than RAP in networks with high loss rates where TCP behaviour is far from AIMD behaviour [17]. Rejaie *et al.* in Reference [11] provide simulation results that tests RAP's TCP-friendliness only. In this paper, we provide results that test IFTP's TCP-friendliness as well as some of the important metrics such as packet delay, delay jitter, packet loss and link utilization.

3. ANALYTICAL MODEL

In this section, we derive the conditions for constructing a rate-based control mechanism for IFTP that gives a faithful emulation of TCP behaviour in all stages of the lifetime of the transport-level connection. We will show later how to modify our model by using a simple scale factor extension to provide better QoS-differentiated services to selected flows.

Consider two concurrent connections in a WAN environment: an unreliable connection running the IFTP transport protocol for real-time media streams and a reliable connection running TCP. The general TCP behaviour used in the design of IFTP can be described as follows. TCP starts its connection in the slow-start phase. It doubles the number of packets in its congestion window (cwnd) every round trip until it reaches a threshold called ssthresh. It then enters the congestion avoidance phase where it increases its window linearly until it reaches the maximum receiver window. If a number of duplicate acknowledgments (ACKs), usually three, are received, TCP assumes that a packet is lost. TCP then reduces the size of the sender's window by half and enters the congestion avoidance phase.

3.1. Inter-packet gap

First we calculate the inter-packet gap for IFTP that makes both connections transmit the same number of packets in the same round trip.

Assume the following parameters, where P is the packet size in bytes, τ the transmission time of a single packet (in s), G_i the inter-packet gap for IFTP, i.e. the time period (in s) between transmitting the last bit of one packet until transmitting the first bit of the next packet. The subscript i represents the adjustment (round trip) period under consideration. Thus in the initial adjustment period, the inter-packet gap is G_0 . In the next period, a gap of the length G_1 is used

to separate successive packets and so on and R the smoothed round trip time (in s) between the sender and the receiver which is also equal to the length of each adjustment period for IFTP. In this paper, we assume a WAN Internet environment and hence $R \gg$ TCP window.

Consider now the i th period and assume that IFTP transmits n packets during this period.

Slow-start phase: The slow-start phase is enforced by TCP as long as no packets are lost and the ssthresh threshold has not been reached. During slow start, TCP doubles the number of packets in the window every round trip. Let n be the number of packets successfully sent in the i th period. Therefore, in the $i + 1$ th period IFTP should send $2n$ packets similar to TCP. Thus, the number of packets that should be sent in the $i + 1$ th round trip period is governed by the following equation:

$$n \times (G_i + \tau) = 2n \times (G_{i+1} + \tau) = R \quad (1)$$

Where $n < \text{ssthresh}$. Thus in the slow-start phase, the inter-packet gap in the period under consideration can be obtained using Equation (2).

$$G_{i+1} = \frac{1}{2} \times (G_i - \tau) = \frac{R}{2n} - \tau \quad (2)$$

Next, we need to define how IFTP exits the slow-start phase. This means we need to choose ssthresh for IFTP to be 'friendly' with TCP. If ssthresh for IFTP is larger than that of TCP then TCP will enter congestion avoidance first, while IFTP is still doubling its bandwidth. We adopt the policy of using a common ssthresh for IFTP and TCP. Both IFTP and TCP can choose ssthresh based on the 'bandwidth * delay' product [18]. In this way, we can guarantee that both IFTP and TCP have fair values of ssthresh.

Congestion avoidance phase: In the congestion avoidance phase, IFTP mimics TCP by increasing the number of packets in each round trip linearly. That is, in the $i + 1$ th period, IFTP should send $n + 1$ packets. Equation (3) shows this relation. From Equation (3), we can calculate the inter-packet gap that should be used during transmission in the congestion avoidance phase as shown in Equation (4).

$$n \times (G_i + \tau) = (n + 1) \times (G_{i+1} + \tau) = R \quad (3)$$

$$G_{i+1} = \frac{n \times G_i - \tau}{n + 1} = \frac{R}{n + 1} - \tau \quad (4)$$

Where n is the number of packets successfully sent in the i th period.

Packet loss: If a packet is lost and three packets after the missing one are received, we would like IFTP to reduce the number of packets by half to roughly emulate the behaviour of TCP after fast recovery. In this case, the number of packets sent in the $i + 1$ th period and the corresponding inter-packet gap can be controlled using Equations (5) and (6), respectively.

$$n \times (G_i + \tau) = \frac{n}{2} \times (G_{i+1} + \tau) = R \quad (5)$$

$$G_{i+1} = 2G_i + \tau = \frac{2R}{n} - \tau \quad (6)$$

From Equation (6), we notice that the inter-packet gap needs to be more than doubled to cut the number of packets by half.

3.2. Enabling QoS-differentiated services in IFTP

With a simple extension, IFTP can provide higher, but controlled, bandwidth to selected best-effort flows. In Section 3.1, we presented the standard mode of IFTP that faithfully emulates the behaviour of TCP. In this section, we present a simple extension that guarantees that the selected flow will attain a scaled (enhanced) value of the bandwidth obtained in the standard mode. A scale factor, S , is used to give non-standard bandwidth allotment to selected best-effort flows throughout the lifetime of the connection. Equations (1)–(6) are used as before to compute the number of packets n and the gap G during each adjustment period. The standard sending rate (which emulates TCP) is then multiplied by a scale factor S , i.e. the number of packets during a round trip is changed from n to S^*n . The modified gap, G_m , is obtained by the following relation:

$$n \times (G + \tau) = S \times n \times (G_m + \tau)$$

and hence

$$G_m = [G - \tau \times (S - 1)]/S$$

Although a value of S smaller than 1 can be used to allocate a degraded bandwidth, we envision that this extension would be mostly used with values of S larger than 1 (e.g. $S = 1.5$ or 2) which would increase the sending rate of the application and provide the capability to offer QoS-differentiated services to selected connections. Notice that the enhanced bandwidth still emulates the general behaviour of a TCP connection, i.e. the sender reduces its sending rate whenever TCP shrinks its window.

4. THE IFTP PROTOCOL

The most essential property of IFTP that makes it different than a non-IFTP such as UDP is that IFTP uses ACKs in order to be compatible with TCP. The ACK scheme for IFTP, however, does not need to be a robust one as that of TCP; a relaxed ACK approximation similar to that of the unreliable RAP protocol [11] would be sufficient for IFTP. Unlike TCP, the IFTP algorithm does not provide any reliability by retransmitting lost packets. IFTP keeps track of ACKs received and round trip times only to adjust its transmission rate in a manner that is friendly to TCP. To do so, IFTP adjusts the value of inter-packet gap (IPG) which is a controllable parameter. Thus, as stated earlier, IFTP can afford to have a crude approximation of the ACK scheme.

The Internet friendly scheme starts by setting n to 1. It sends one packet and waits for its ACK to arrive to set the initial value of the round trip time (RTT). The implementation has four main parts:

1. Slow start.
2. Congestion avoidance.
3. Packet loss.
4. Timeout.

In the **slow-start** phase, the IFTP sender doubles the number of packets to be transmitted every RTT and updates G as calculated in Equation (2). The sender does that when all the ACKs of the packets transmitted in the previous RTT are received. The sender uses the variable

ack_count to keep track of the number of ACKs received in the current RTT. The IFTP sender exits the slow-start phase when n reaches a threshold (sssthresh). It then enters the **congestion avoidance** phase where it increases n linearly every RTT. It updates G according to Equation (4). This phase continues until n reaches the receiver's maximum advertized window. The sender uses the routine Transmit_window to send the required number of packets in each round trip. Transmit_Window is a thread that executes in parallel once it is activated from the main loop to perform one sweep for the transmission of the n packets of the new window.

Unlike a TCP receiver, an IFTP receiver does not use cumulative ACKs. An IFTP receiver merely transmits the sequence number of the packet it receives. If the sender detects a gap in the sequence number of the ACKs received, it assumes that a packet or more has been lost. This is an indication that the network is congested. When the number of ACKs received by the sender after the missing one reaches a certain threshold (aftermissing_thresh) it reacts as follows:

1. The IFTP sender halves its ssthresh or sets it to 2 whichever is larger to reduce the amount of data.
2. It halves the number of packets to be transmitted in the next RTT.
3. It sets the inter-packet gap according to Equation (6). The IFTP reacts only once upon loss detection. Subsequent ACKs received in the same round trip indicating further packet loss are ignored. The sender exits the packet loss phase when it receives an ACK with a sequence number greater than or equal to the last one it transmitted before the packet loss phase.

If for any reason no ACKs are received for a calculated period of time (timeout period), the IFTP sender uses its timeout mechanism. It drops ssthresh to half the current transmission rate. It then resets n to 1 and G to $R - \tau$ where τ is the transmission time of one packet. A window of size 1 is subsequently transmitted. Note that in the packet loss and timeout phases **no retransmission of packets take place**. Upon detection of a missing packet in either phase, IFTP reduces its transmission rate but transmits new packets. It does not keep track of which or how many packets have been lost. The bandwidth used by TCP and IFTP will be roughly the same. However, the bandwidth of TCP is expected to be lower than that of IFTP. This is because TCP retransmits all lost packets and waits (in some implementations) until it receives ACKs confirming that all lost packets have been received.

To calculate the timeout period, IFTP uses a mechanism similar to that used in TCP. Four variables are used to estimate the roundtrip time and set the timeout timer [19]: rtt, srtt, rttvar and backoff. The variable rtt is the measured round trip time which could change from one packet to the other depending on the level of congestion. The variable srtt is the smoothed (i.e. average) value of rtt while rttvar is a measure of the variability of rtt. We referred to srtt in the equations of Section 3 by R for simplicity. Round trip time sample arrives with new ACKs. The rtt sample is computed as the difference between the current time and a time field in the ACK packet, which is equal to the time the packet was issued. When the first sample is taken, its value is used as the initial value for srtt. Half the first sample is used as the initial value for rttvar. For subsequent samples, the values are updated as follows:

$$\text{srtt} = \frac{7}{8} \times \text{srtt} + \frac{1}{8} \times \text{rtt} \quad (7)$$

$$\text{rtt var} = \frac{3}{4} \times \text{rtt var} + \frac{1}{4} \times \text{rtt} - \text{srtt} \quad (8)$$

The *backoff* is initially 1. The timeout timer is set to

$$\text{timeout timer} = \text{current time} + \text{backoff} \times (\text{srtt} + 4 \times \text{rtt var} + 1) \quad (9)$$

The *backoff* factor doubles each time a timeout occur to a maximum of 64 (Karn's exponential timer backoff) [20]. The timer granularity in IFTP is set to the nearest millisecond, as this is usually the case in TCP.

Figure 1 shows the pseudocode for the standard mode of IFTP (i.e. the extension given in Section 3.2 is not included in this code but can be added straightforwardly). Unlike TCP, IFTP does not maintain any window nor retransmit any lost packets. The variable 'next' which points to the sequence number of the packet to be transmitted is always advancing forward and is never set back to retransmit a window. In IFTP, fast recovery and timeouts translate into changing the rate of transmission by changing the inter-packet gap without changing the monotonously increasing order of the sequence number of the packet to be transmitted.

5. BANDWIDTH REQUIREMENT

In this section, we calculate the bandwidth of the standard mode of IFTP in three cases: an ideal non-congested communication environment that does not suffer any bit error rate or lost frames. This provides an easy to calculate upper bound for the bandwidth of IFTP. The second case is a non-ideal environment in which some packets are lost. This is the general case. The third is a worst-case environment in which the IFTP feedback mechanism is not functioning at all (i.e. ACKs from the receiver are lost and are never seen by the sender).

Unlike TCP, IFTP is not a window-based protocol. The sender does not maintain a transmission window to buffer transmitted packets for possible retransmission. Rather, IFTP is a rate-based scheme which seeks to be conformant with TCP behaviour. The sender controls its transmission rate by adjusting the inter-packet gap and lost packets are not retransmitted (best-effort multimedia streaming applications). However, a window-like action is created in IFTP as a result of the periodic adjustment of the inter-packet gap. The number of packets transmitted within each adjustment period of IFTP is logically equivalent to the window size of TCP. We will therefore refer to this number as the window size of IFTP and denote it by W .

5.1. Ideal case

Figure 2 shows an example of the window size versus time for an ideal IFTP connection with $W_s = 32$ and $W_{\max} = 50$. The three time periods shown in this graph correspond to the three phases: slow start, congestion avoidance and maximum window as explained earlier.

Let θ be the bandwidth of ideal IFTP (bits/s), N the total number of packets transmitted, T the total transmission time (s), W_s ssthresh (in packets), assume it is a power of 2, W_{\max} the maximum window size (in packets), R the round trip time which is the basis of the adjustment period for IFTP (s), T_1 the length of time of the slow-start phase (s), T_2 the length of time of the congestion avoidance phase (s), T_3 the length of time when the window is of maximum size (s) and N_i the number of packets transmitted in T_i where $i = 1, 2, 3$.

Note that the window size W during the slow-start phase is $1 < W < W_s$, during the congestion avoidance phase is $W_s < W \leq W_{\max}$, and during the maximum size phase is $W = W_{\max}$.

```

//Initialization
n=1 // number of packets transmitted in current period
una=1 // first unacked packet
next=1 // next packet to be transmitted
prev_last=0 // last packet transmitted before dupliacte acks

Transmit_window() //initial window is of size n=1
Wait for ACK message
//If ACK does not arrive, a timeout interrupt will eventually cause the Timeout() routine to excute. If ACK
//arrives, set ackIn.
ackIn = Sequence # of the acknowledged packet

//main processing phase
if ( ackIn > una ) {
    // packet loss
    after_missing++
    if(after_missing == after_missing_thresh) {
        ssthresh = max (ssthresh/2,2)
        n=n/2
        G= (srtt/n) -  $\tau$ 
        prev_last= next -1
        Transmit_Window()
    }
    if ( after_missing > after_missing_thresh && ackIn >= prev_last)
        una =ackIn+1 //exit pkt loss phase
    if(ack_count >0) ack_count=0
}
else { after_missing=0
ack_count++
if(ack_count ==n) {
    //received ACKs for all pkts transmitted in previous window
    ack_count=0
    if (n < ssthresh) {
        //slow start phase

        n = 2*n
    }
    else if (n < W_max) {
        //congestion avoidance phase
        n= n+1
    }
    else { n and G stay the same}
    G= (srtt/n) -  $\tau$ 
    Transmit_Window()
}
una= ackIn +1
}

Transmit_Window()
last= next + n
while (next <= eof && next < last)
{
    send packet
    Pause (inter-packet gap)
    next ++
}

Timeout:
//This is an interrupt-driven routine which is invoked if the timeout timer expires while waiting for ACK
If (timeout timer expired ) {
    ssthresh = max (n/2, 2)
    n=1
    G= srtt -  $\tau$ 
    Transmit_Window()
    una=next - 1
}

```

Figure 1. Pseudocode for IFTP.

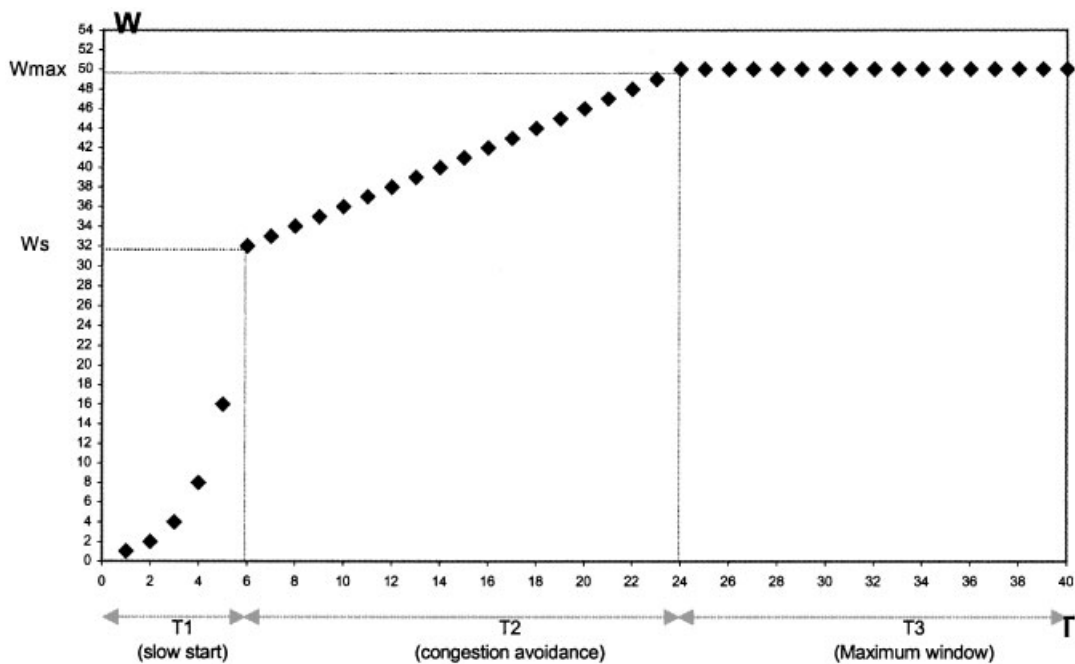


Figure 2. Example of Window size versus time for ideal IFTP.

The bandwidth θ is given by

$$\theta = \frac{N}{T} \times P \quad (10)$$

where P is the size of packets in bits and

$$T = T_1 + T_2 + T_3 \quad (11)$$

It is easy to calculate T_1 , T_2 and T_3 . They are equal to the number of packets in each phase multiplied by RTT. In the slow start phase, the window starts with a value of 1 then it is doubled every RTT until it reaches W_s . Thus the number of adjustment periods in T_1 is equal to $\lg W_s + 1$, where \lg denotes base 2 logarithm. Thus T_1 is equal to

$$T_1 = (\lg W_s + 1) \times R \quad (12)$$

In the congestion avoidance phase, the window is increased by one every round trip until it reaches W_{\max} , thus T_2 is equal to

$$T_2 = (W_{\max} - W_s) \times R \quad (13)$$

In T_3 , the window is of size W_{\max} and is constant until the end of the file is reached. In other words, during T_3 a constant minimum inter-packet gap is used to separate packets. Recall that we are analysing the ideal case in which no packet loss occurs. Thus T_3 is equal to

$$T_3 = \frac{(N - N_1 - N_2)}{W_{\max}} \times R \quad (14)$$

The number of packets transmitted in each phase can be easily calculated as follows:

$$N_1 = 2^{\lg W_s + 1} - 1 = 2 \times W_s - 1 \quad (15)$$

$$N_2 = (W_{\max} - W_s - 1) \times \left(\frac{W_{\max} + W_s}{2} \right) \quad (16)$$

Short files containing $N \leq N_1$ packets are completely transmitted during the slow-start phase. Files with $N \leq N_1 + N_2$ packets are completely transmitted before the end of the congestion avoidance phase. The bandwidth analysis is given below.

Case 1 ($N \leq N_1$): This case corresponds to short files that are completely transmitted before the end of the slow-start phase. The sender starts with a window of size 1 packet and then doubles its window after each round trip. The exponential growth of the window in this phase leads to the following value of bandwidth.

$$\theta(N) = \frac{N \times P}{\lceil \lg N \rceil + 1 \times R} \quad (17)$$

where $\lceil x \rceil$ is the smallest integer greater than x .

Case 2 ($N_1 < N \leq N_1 + N_2$): This case corresponds to files completely transmitted before the end of the congestion avoidance phase. After reaching the threshold window size W_s at the end of the slow-start phase, the sender increases the window size by 1 packet in each round trip. Thus, the successive window sizes in the slow-start phase are 1, 2, 4, ..., W_s while those in the congestion avoidance phase are $W_s + 1$, $W_s + 2$, etc. This leads to the following equation for bandwidth where the first term ' $\lg(W_s) + 1$ ' of the denominator corresponds to the number of round trips of the completed slow-start phase and the second term (solution of a quadratic equation) is the number of round trips in the ongoing congestion avoidance phase.

$$\theta(N) = \frac{N \times P}{\left[(\lg W_s + 1) + \left\lceil \frac{\sqrt{(2W_s + 1)^2 + 8(N - 2 \times W_s + 1) - (2W_s + 1)}}{2} \right\rceil \right] \times R} \quad (18)$$

Case 3 ($N > N_1 + N_2$): This case corresponds to arbitrarily longer files that continue to be transmitted after the congestion avoidance phase increases the window to the maximum value W_{\max} . Notice that the number of round trips in the completed congestion avoidance phase is simply $W_{\max} - W_s$. The equation for bandwidth has a similar logic to that of Equation (18), but the denominator has three terms corresponding to the slow start, congestion avoidance phase and the maximum window (steady-state) phase

$$\theta(N) = \frac{N \times P}{\left[(\lg W_s + 1) + (W_{\max} - W_s) + \left\lceil \frac{1}{W_{\max}} \left[N - 2^{\lg W_s + 1} + 1 - (W_{\max} - W_s - 1) \times \left(\frac{W_{\max} + W_s}{2} \right) \right] \right\rceil \right] \times R} \quad (19)$$

for very long files (i.e. $N \rightarrow \infty$), the bandwidth is obtained from Equation (19) by applying L'Hopital's rule:

$$\lim_{N \rightarrow \infty} \theta = \frac{W_{\max} \times P}{R} \quad (20)$$

Note that the steady-state bandwidth given in Equation (13) is only achievable if it is less than the maximum link bandwidth.

5.2. Non-ideal case

In this section, we calculate a non-ideal steady-state bandwidth for IFTP under the model used by Floyd and Fall in Reference [3]. This model assumes an environment with the following properties:

- p is the average packet drop rate.
- p is non-bursty in the sense that only one packet can be dropped in the same adjustment period (i.e. same window).
- A packet can be dropped only when the number of packets transmitted in the period is equal to n .

The packet drop probability can be represented by a steady-state cycle in which IFTP decreases the number of transmitted packets by half and then linearly increases them every round trip until they reach n . The number of successfully transmitted packets before a packet is dropped, in the steady-state cycle is given by

$$\frac{n}{2} + \left(\frac{n}{2} + 1\right) + \left(\frac{n}{2} + 2\right) + \dots + n = \frac{3}{8}n^2 + \frac{3}{4}n \quad (21)$$

The average number of transmitted packets per window is given by

$$E(N_{\text{window}}) = \frac{3}{4}n \quad (22)$$

Therefore, in a steady-state cycle the upper bound of the bandwidth θ_{ss} can be calculated as follows:

$$\theta_{\text{ss}} \leq \frac{0.75 \times n \times P}{R} \quad (23)$$

Thus, the resulting TCP-friendly IFTP will give the same sending rate under the steady-state congestion model defined in Appendix B of Reference [3]. Note that n is the maximum window size attained in Floyd's cyclic model. Whenever that window size is reached, a packet loss occurs and the window shrinks to $n/2$.

We can also calculate the steady-state bandwidth in terms of the packet drop rate p . From (21)

$$p = \frac{1}{(3/8)n^2 + (3/4)n} \quad (24)$$

From (24),

$$n = \sqrt{1 + \frac{8}{3p}} - 1 \quad \text{or} \quad n \leq \sqrt{\frac{8}{3p}} \quad (25)$$

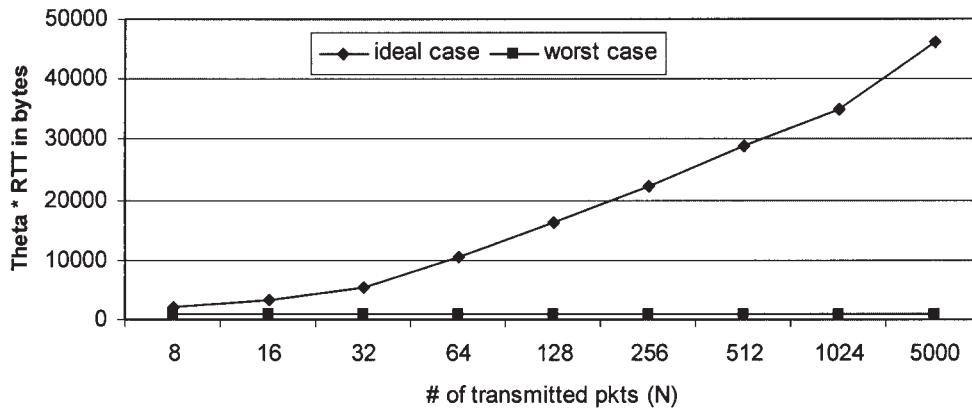


Figure 3. Bandwidth \times RTT for ideal and worst cases.

Substituting for n from (25) into (23), we get

$$\theta_{ss} \leq \frac{1.5 \times \sqrt{2/3} \times P}{R \times \sqrt{P}} \quad (26)$$

5.3. Worst case

In this case, we assume that for some worst-case reason all ACKs from the receiver are lost. This provides a lower bound on the performance of IFTP. The IFTP sender then uses a timeout mechanism that allows it to continue its transmission. As was explained in Section 4, IFTP calculates its timeout period in a fashion similar to that employed in TCP implementations. Using Equation (9) from Section 4, in the worst case the value of backoff will be equal to 64, while s_{rtt} , $rttvar$ will be equal to 0. Thus the timeout timer will be equal to 64 ms. Every time the timeout timer expires, IFTP will transmit one packet. Therefore, the bandwidth will be equal to

$$\theta = \frac{P}{\tau + 0.064} \text{ bits/s} \quad (27)$$

Figure 3 shows the bandwidth \times RTT for ideal and worst cases. The x -axis in Figure 3 represents the values for N (the number of packets in the file). The range $N = 8$ –32 corresponds to the slow-start phase (case 1), $N = 64$ –512 corresponds to the congestion avoidance phase (case 2), while $N = 1024$ –5000 corresponds to the maximum window phase (case 3).

6. SIMULATION

In this section, we first illustrate the problem of unfairness by simulating the network topology of Figure 4 without the IFTP sources and sinks. The routers have a single output queue for each attached link and use FCFS scheduling and drop-tail queuing. We use Reno TCP which is widely used on the Internet. We use packets of size 1000 bytes for both TCP and UDP. ACK sizes for TCP are equal to 40 bytes. The total path delay of 11 ms for UDP is the same as TCP. The bottleneck buffer is of size 50 packets. There is an infinite amount of data for UDP and

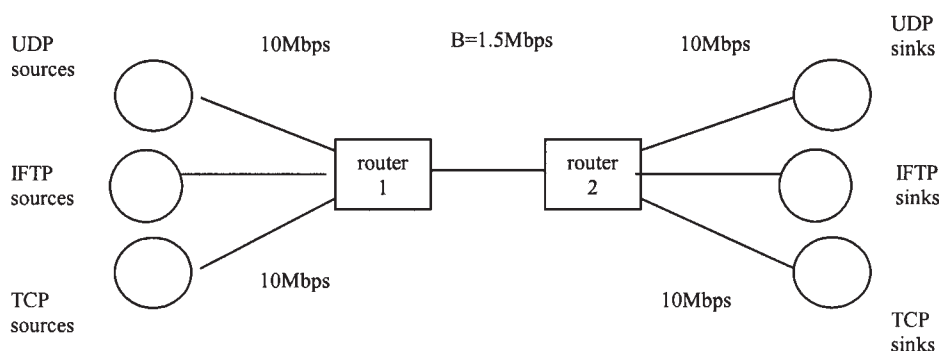


Figure 4. Simulation network.

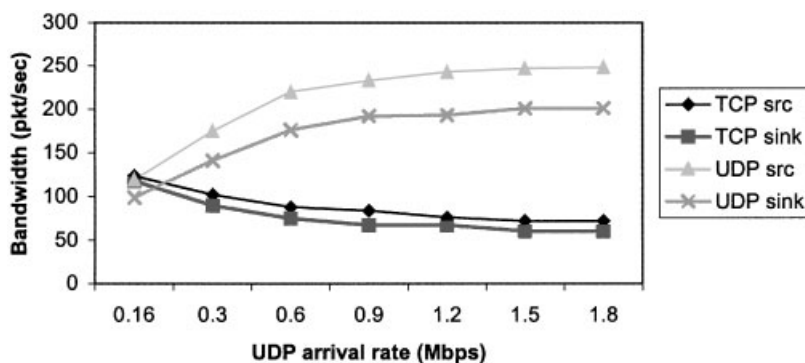


Figure 5. Illustration of non-TCP friendliness by simulation.

TCP. We vary the sending rate of UDP for up to 1.8 Mbps, which is larger than the bandwidth of the bottleneck link. The results are illustrated in Figure 5.

When the sending rate of UDP is small, TCP has higher bandwidth than UDP. Also the bandwidth of TCP at the source is almost equal to that of its sink. This means that TCP almost does not suffer any losses. As the sending rate of UDP increases, the bandwidth of UDP increases, stealing more bandwidth from TCP, while TCP backs off in response to packet losses. This results in unfairness to TCP because it competes with unresponsive UDP for bandwidth under FCFS scheduling. Obviously, transport protocols like UDP are '*non-TCP friendly*'.

Besides this problem, there is another crucial problem we noticed through our simulations. As the arrival rate of UDP increases, not only it deprives responsive TCP from its bandwidth, but also the difference between the bandwidth of UDP at the source and that at the sink increases. This is shown in Figure 5. This result is due to the increase of packet loss at the routers as a result of congestion. Unlike UDP, TCP maintains the number of packets lost to a minimum due to its reliable nature even though its bandwidth decreased.

It is important to note here that the reduction of TCP bandwidth starts when the arrival rate is only 0.3 Mbps because the bottleneck buffer was limited to 50 packets.

6.1. TCP friendliness

In this section, we use simulation to demonstrate the most important property of IFTP: TCP-friendliness. In this experiment, we used two IFTP sources and two TCP sources (no UDP sources were used) sharing the bottleneck link of bandwidth $B = 1$ Mbps. The total path delay of 11 ms for IFTP is the same as TCP. We used the IFTP parameters to be the same as TCP unless otherwise stated. That is

Packet size = 1000 bytes
 ACK size = 40 bytes
 ssthresh = 32
 W_max = 50
 dupack_thresh = aftermissing_thresh = 3

We ran the simulation for 120 s. We calculated the average bandwidth of TCP sources and IFTP sources, respectively. The results are shown in Figure 6. This figure shows that IFTP and TCP both have a fair almost equal share of bandwidth. Even though this figure shows that the average bandwidth of TCP is higher than that of IFTP, it is not always the case. There are some scenarios in which IFTP has higher bandwidth than TCP as shown later in Table IV of Section 6.4. Since FCFS scheduling and drop tail queuing are used there is no guarantee on which IFTP sources or TCP sources will suffer from packet loss. If a packet loss occurs it is not guaranteed whether the lost packet will cause which source to suffer timeouts.

Table I shows the value of the average value of G (the inter-packet gap) for two IFTP sources over the period of 120 s. It also shows the standard deviation (jitter) from this average. IFTP was adaptive to the network requirements and changed G accordingly. Unlike UDP in Figure 5 where the sending rate was not controlled thus caused the deprivation of the TCP bandwidth.

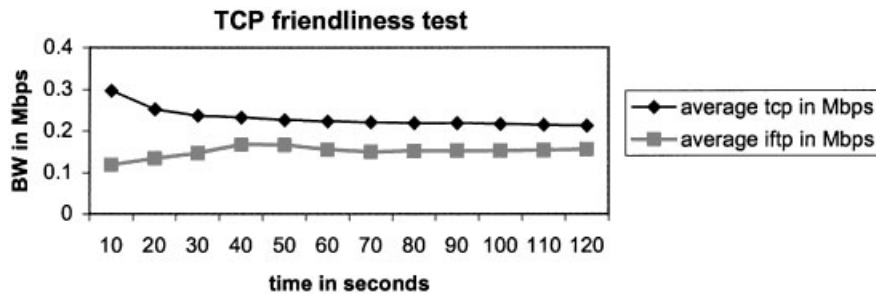


Figure 6. TCP friendliness test.

Table I. Values of inter-packet gap (G) and its standard deviation for IFTP sources.

	IFTPO	IFTPI
G in ms	22.98	23.75
stdev(G) in ms	4.01	4.86

6.2. Packet loss

In this section, we use simulation to test another performance metric, which is the number of packets lost. For the tests in this section and Section 6.3, we had 1 UDP source, 1 TCP source and 2 IFTP sources. The bandwidth of the bottleneck link was $B = 1.5$ Mbps. All other parameters are the same as Section 6.1. We varied the arrival rate of UDP over the range of 1.8–0.16 Mbps. Table II shows the number of packets lost per source for UDP, IFTP and TCP. The top row shows the arrival rate of UDP in Mbps.

We can notice the tremendous difference between the number of packets lost for UDP and those for IFTP. UDP can lose one or two orders of magnitude of packets more than IFTP. Of course the higher the arrival rate the more the number of packets lost for UDP. IFTP loses packets in the same range as TCP. However, we cannot guarantee a certain number of packet drop with FCFS scheduling. IFTP and TCP both are rate-controlled protocols thus they react to congestion. This keeps the number of their lost packets to a minimum. UDP and IFTP do not use retransmission methods. Therefore, it is a good feature for IFTP to keep the number of lost packets low. UDP applications on the other hand have no way to control the arrival rate at times of congestion, thus can end up losing lots of packets, and worsen the congestion.

Note that in this scenario there are packets lost even when UDP arrival rate is only 0.16 Mbps due to the limited size buffer at the bottleneck router.

6.3. Delay and delay jitter

We define the delay to be the time a packet is issued at the source till the time it is received. The delay jitter is the standard deviation of the delay. Table III shows the average delay and delay jitter for UDP and IFTP over the period of 120 s for the arrival rate range of UDP as above. From Table III we can notice that the average delay of IFTP is almost half that of UDP. Also the delay jitter of IFTP is less than that of UDP. This is because in case of UDP the packets suffer long queuing delays at the router especially as the sending rate increases.

6.4. Effect of bottleneck bandwidth and delay

We studied the effect of bottleneck bandwidth and delay [9]. We changed the delay: 3 ms and 3 ms with bandwidths 0.5, 1.0 and 1.5 Mbps. Table IV shows the average TCP bandwidth (top

Table II. Number of packets lost for UDP versus IFTP sources.

	1.8	1.2	0.9	0.6	0.3	0.16
UDP	3760	3573	3353	2882	1545	238
IFTP	16	30	16	25	15	11
TCP	16	16	17	25	28	12

Table III. Delay and delay jitter for IFTP versus UDP.

	IFTP	UDP
Delay in ms	541.05	990
Delay jitter in ms	105.42	125.45

Table IV. Link utilization and average bandwidths of TCP and IFTP for different bottleneck bandwidths (B) and delay (D).

	$B = 0.5$ Mbps	$B = 1.0$ Mbps	$B = 1.5$ Mbps
$D = 0.003$ ms	0.2	0.45	0.207
	0.06	0.05	0.38
	100%	100%	100%
$D = 3$ ms	0.19	0.212	0.41
	0.03	0.156	0.07
	86%	73.4%	60%

row) in Mbps, the average IFTP bandwidth (middle row) in Mbps and the utilization of the link (bottom row). We notice that when $B = 1.5$ and $D = 0.003$ ms and when $B = 1.0$ and $D = 3$ ms, TCP and IFTP had relatively close average bandwidths. Otherwise TCP had a higher bandwidth than IFTP. In those cases, IFTP happened to timeout several times (4–5) which resulted in its low bandwidth. The utilization of the link ranged from 60 to 100%. This is similar to the results in Reference [9] in their tests of the direct adjustment algorithm against TCP. TCP is inherently oscillatory [21]. Hence, TCP and consequently IFTP, as it is an emulation of TCP, do not converge to an equilibrium state but oscillates around the optimal state. The variance in the achieved utilization results from the oscillatory behaviour of TCP and IFTP.

7. DATAGRAM CONGESTION CONTROL PROTOCOL (DCCP)

The datagram congestion control protocol (DCCP) is a recently proposed unreliable transport protocol incorporating end-to-end congestion control [22]. DCCP has been proposed well after the submission of the original draft of this paper; at the time of writing the revised final version of this paper, DCCP is still a work in progress proposed in recent Internet Drafts. In this section, we provide some general comments about DCCP in relation to the IFTP protocol proposed in this paper.

DCCP has been motivated by the need for a new transport protocol that offers unreliable delivery; accommodates alternative congestion control algorithms; accommodates the use of explicit congestion notification (ECN); and requires minimal overhead in packet size and in the state and CPU processing required at the data endpoints.

DCCP allows the use of different congestion control mechanisms on each direction of a connection [23]. Each conformant congestion control mechanism is assigned an 8-bit congestion control identifier, or CCID (a number from 0 to 255). CCID of value 2 is reserved for TCP-like congestion control which uses additive increase, multiplicative decrease (AIMD) congestion control with behaviour modelled directly on TCP. CCID 3 is reserved for the receiver-based TCP-friendly rate control (TFRC) method that tries to minimize abrupt changes in the sending rate, while maintaining longer-term fairness with TCP. Our proposed IFTP is basically a conformant protocol that can be used in the DCCP model. IFTP satisfies the desire of DCCP in minimizing the state maintained by the data sender and in achieving minimal overhead. IFTP would complement the congestion control methods offered by DCCP and would provide a capability of QoS-differentiated services through its extended mode (Section 3.2). The generic DCCP packet header can be used to encapsulate IFTP and DCCP's reliable negotiation

mechanism is a welcome facility that would be useful in setting up IFTP connections and in negotiating the scale factor of IFTP's extended mode. The implementation of IFTP would much benefit from the various options offered by DCCP; For example, the connection nonce option would help in restoring proper operation if the endpoints get out of sync.

Finally, we believe that there should be no major difficulty in having an IFTP implementation within DCCP that is fully ECN-aware. In particular, ECN Nonces would be particularly useful to IFTP in that it would help prevent loss cheating. In this type of cheating, the receiver (a user of a media streaming or internet telephony application) would falsify ACK information by reporting that dropped packets were actually received. The sender (media streaming server) would not therefore react to congestion and the receiver may continue to enjoy better quality at the expense of TCP and honest TCP-friendly connections.

8. CONCLUSION

In this paper, we proposed a design of an IFTP for media streaming whose goal is to contribute to the good health of the Internet and avoid starving other good behaving protocols like TCP. We developed an analytical model to derive the conditions for the congestion control mechanism of IFTP. We presented a simple extension that allows offering QoS-differentiated services to selected connections. In addition to designing a generalized framework of the IFTP algorithm, we also derived tight upper bounds of the amount of bandwidth that IFTP may be able to claim from TCP. We calculated analytically the bandwidth of IFTP in ideal, non-ideal, and worst-case scenarios. We evaluated IFTP through simulation. We tested fairness and important performance metrics namely, packet loss, delay, delay jitter and link utilization. Our simulations have shown that IFTP is TCP friendly. It also provides less packet loss, delay and delay jitter than UDP. The simulation tests have also shown that IFTP maintains its design goals of TCP-friendliness for various link bandwidths and delays.

ACKNOWLEDGEMENTS

This work has been partially supported by NSF under grant 0086251 and by a grant from ARO. The views and conclusions herein are those of the authors and do not represent the official policies of the funding agencies or the University of Central Florida. The authors are grateful to the three anonymous reviewers for their comments and suggestions that helped improve the quality of this paper.

REFERENCES

1. Jacobson V, Karel M. Congestion avoidance and control. *ACM SIGCOMM* 1988; 314–329.
2. ElAarag H, Bassiouni M. Performance evaluation of TCP connections in ideal and non-ideal network environments. *Journal of Computer Communications* (Elsevier: Amsterdam) 2001; **24**(18):1769–1779.
3. Floyd S, Fall K. Promoting the use of end-to-end congestion control in the internet. *IEEE/ACM Transactions on Networking* 1999; **7**(4):458–472.
4. Mahdavi J, Floyd S. TCP-friendly unicast rate-based flow control. *Technical note sent to end2end-interest mailing list*, http://www.psc.edu/networking/papers/tcp_friendly.html, January 1997.
5. Crowcroft J, Oechslein P. *Differentiated End-to-End Internet Services using a Weighted Proportional Fair Sharing TCP*. June 1998.
6. Shenker S. Making greed work in networks: a game-theoretic analysis of switch service disciplines. In *Proceedings of the SIGCOMM Symposium on Communications Architectures and Protocols*, August 1994; 47–57.

7. Legout A, Biersack EW. Beyond TCP-friendliness: a new paradigm for end-to-end congestion control. *Eurecom Technical Report*, available at URL <http://www.eurecom.fr/~legout/research/research.html>, October 1999.
8. Legout A, Biersack EW. PLM: fast convergence for cumulative layered multicast transmission schemes. In *ACM SIGMETRICS Performance Evaluation Review, Proceedings of the 2000 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, vol. 28(1), June 2000.
9. Sisalem D, Schulzrinne H, Emanuel F. The direct adjustment algorithm: a TCP-friendly adaptation scheme. *Technical Report, GMD-FOKUS*, August 1997. Available from <http://www.fokus.gmd.de/usr/sisalem;http://citeseer.nj.nec.com/sisalem97direct.html>
10. Sisalem D, Schulzrinne H. The loss-delay based adjustment algorithm: A TCP-friendly adaptation scheme. *Workshop on Network and Operating Systems Support for Digital Audio and Video*, July 1998.
11. Rejaie *et al.* RAP: an end-to-end rate-based congestion control mechanism for real time streams in the internet. *IEEE Infocom*, 1999.
12. Rejaie *et al.* Quality adaptation for congestion controlled video playback over the internet. *ACM SIGCOMM*, September 1999.
13. Widmer J, Denda R, Mauve M. A survey on TCP-friendly congestion control. *IEEE Network* 2001; **15**(3):28–37.
14. Sisalem D, Wolisz A. LDA + TCP-friendly adaptation: a measurement and comparison study. In *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'2000)*, Chapel Hill: NC, USA, June 25–28, 2000.
15. Floyd S, Handley M, Padhye J, Widmer J. Equation-based congestion control for unicast applications. *ACM SIGCOMM Computer Communication Review, Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, vol. 30(4), August 2000.
16. Rhee I *et al.* TEAR: TCP emulation at receivers—flow control for multimedia streaming. *Technical Report*, Department of Computer Science: NCSU, April 2000.
17. ElAarag H, Bassiouni M. Transport control protocols for wireless connections. *Proceedings of IEEE Vehicular Technology Conference (VTC99)*, Houston, TX, May 1999; 337–341.
18. Hoe JC. Improving the start-up behaviour of a congestion control scheme for TCP. *ACM SIGCOMM Computer Communication Review, Conference Proceedings on Applications, Technologies, Architectures, and Protocols for Computer Communications*, vol. 26(4), August 1996.
19. Fall K, Floyd S. Simulation based comparisons of Tahoe, Reno, and SACK TCP. *ACM Computer Communication Review* 1996; **26**(3):5–21.
20. Comer D. *Internetworking with TCP/IP*, vol. 1–3. Prentice-Hall: Englewood Cliffs, NJ, 1991.
21. Chiu D, Jain R. Analysis of the increase/decrease algorithms for congestion avoidance in computer networks. *Journal of Computer Networks and ISDN* 1989; **17**(1):1–14.
22. Floyd S, Handley M, Kohler E. Problem Statement of DCCP. *Internet Engineering Task Force-INTERNET DRAFT*, 26 August 2002 (<http://www.icir.org/kohler/dccp/draft-floyd-dcp-problem-01.txt>).
23. Kohler E, Handley M, Floyd S, Padhye J. Datagram congestion control protocol (DCCP). *Internet Engineering Task Force-INTERNET DRAFT*, 30 June 2002 (<http://www.icir.org/kohler/dccp/draft-kohler-dcp-04.txt>).

AUTHORS' BIOGRAPHIES

Mostafa A. Bassiouni received his BSc and MSc degrees in Computer Science from Alexandria University and received PhD degree in Computer Science from the Pennsylvania State University, University Park, in 1982. He is currently a professor of Computer Science at the University of Central Florida, Orlando. His research interests include distributed systems, computer networks, real-time protocols and concurrency control. He has authored some 140 papers published in various computer journals, book chapters and conference proceedings. His research has been supported by grants from ARO, ARPA, NSF, STRICOM, PM-TRADE, CBIS, Harris, and the State of Florida. He is a member of IEEE and ACM.

Hala ElAarag received her BSc and MSc degrees in Computer Science from Alexandria University and received her PhD degree in Computer Science from the University of Central Florida, Orlando, in 2001. She is currently an assistant professor of computer science at Stetson University, DeLand, FL. Her research interests include network protocol architecture, congestion control and performance analysis. She is a member of IEEE and IEEE women in Engineering.